International Journal of Artificial Intelligence,
ISSN 0974-0635; Int. J. Artif. Intell.
2016, Volume 14, Issue Number 1

A Comparison Study of Cooperative Q-learning Algorithms for Independent Learners

Bilal H. Abed-Alguni[1+], David J. Paul[2*], Stephan K. Chalup[3+] and Frans A. Henskens[4+]

[+]School of Electrical Engineering & Computer Science, The University of Newcastle, Callaghan NSW 2308, Australia

[*]School of Science and Technology, The University of New England, Armidale NSW 2351, Australia

[1]Bilal.Abedalguni@uon.edu.au [2]David.Paul@une.edu.au

[3]Stephan.Chalup@newcastle.edu.au [4]Frans.Henskens@newcastle.edu.au

ABSTRACT

Cooperative reinforcement learning algorithms such as BEST-Q, AVE-Q, PSO-Q, and WSS use Q-value sharing strategies between reinforcement learners to accelerate the learning process. This paper presents a comparison study of the performance of these cooperative algorithms as well as an algorithm that aggregates their results. In addition, this paper studies the effects of the frequency of Q-value sharing on the learning speed of the independent learners that share their Q-values among each other. The algorithms are compared using the taxi problem (multi-task problem) and different instances of the shortest path problem (single-task problem). The experimental results when learners have equal levels of experience suggest that sharing of Q-values is not beneficial and produces similar results to single agent Q-learning. However, the experimental results when learners have different levels of experience suggest that most of the cooperative Q-learning algorithms perform similarly, but better than single agent Q-learning, especially when Q-value sharing is highly frequent. This paper then places Q-value sharing in the context of modern reinforcement learning techniques and suggests some future directions for research.

Keywords: Cooperative Learning, Q-learning, Q-values, Q-value Sharing Strategy, Single-agent System, Multi-agent System.

Computing Classification System (CCS): k.3.2

1 Introduction

Reinforcement Learning (RL) is a machine learning technique based on trial and error (Sutton and Barto, 1998; Gilles and Peroumalnaik, 2008). One of the best known RL algorithms is Q-learning (Watkins, 1989; Watkins and Dayan, 1992; Zhang, Mao, Liu and Liu, 2013). This algorithm finds mappings from state-action pairs to values using dynamic programming. These values are known as Q-values, and are calculated using a utility function called a Q-function. The Q-function returns the expected utility of taking a given action in a given state and following a fixed policy after that. The collection of Q-values for all state-action pairs is called a Q-table.

Sharing of Q-values between reinforcement learners is known to accelerate the learning process of individual learners in multi-agent systems (Iima, Kuroe and Emoto, 2011; Eshgh and Ahmadabadi, 2002; Galindo-Serrano, Giupponi, Blasco and Dohler, 2010). Cooperative Q-learning normally takes place in two stages. First, the individual learning stage, where each learner independently uses its own Q-learning algorithm to improve its solution. Second, the learning by interaction stage, in which a Q-value sharing strategy is implemented. A Q-value sharing strategy allows the independent learners to share their Q-values and use this information to obtain new Q-tables.

Some of the best known cooperative Q-learning algorithms are BEST-Q, AVE-Q, PSO-Q (Iima and Kuroe, 2006; Iima and Kuroe, 2007; Iima and Kuroe, 2008; Iima, Kuroe and Matsuda, 2010; Di Mario, Talebpour and Martinoli, 2013; Doğan and Ölmez, 2015), and WSS (Cunningham and Cao, 2012; Pakizeh, Palhang and Pedram, 2013; Hwang, Jiang and Chen, 2015; Ahmadabadi and Asadpour, 2002; Ahmadabadi, Imanipour, Araabi, Asadpour and Siegwart, 2006). Until now there have been no studies that compare the performance of these algorithms and study the effect of the frequency of Q-value sharing on the learning speed. This paper presents a comparison study of these algorithms as well as an algorithm that aggregates their results. The study compares the results in two cases. In case one, the learners have equal levels of experience, and in case two, the learners have different levels of experience. In addition, this paper studies the effects of the frequency of Q-value sharing on the learning speed. For example, does sharing of Q-values after each learning episode have benefits over sharing after each hundred learning episodes?

The remainder of the paper is structured as follows: Section 2 describes cooperative Q-learning, Section 3 discusses some modern approaches to RL, Section 4 discusses the performance of cooperative Q-learning algorithms for the shortest path problem and the taxi problem, Section 5 is a critical analysis of the experimental results and Section 6 is the conclusion of the paper.

2 Cooperative Q-learning

This section describes the two learning stages of cooperative Q-learning in multi-agent systems for BEST-Q, AVE-Q, PSO-Q, WSS, and an algorithm that aggregates their results. The first stage of cooperative Q-learning is similar for each algorithm where each learner implements single agent Q-learning to improve its own solution, while the second stage is different where each algorithm follows a different Q-value sharing strategy.

2.1 First Stage: Q-learning

In the first stage, each learner independently performs Q-learning. The problem model of the Q-learning algorithm is a Markov Decision Process (MDP) that consists of an agent, a set of states S, and a set of actions A_i for each state $s_i \in S$ (Campbell, Givigi and Schwartz, 2015; Abed-alguni, Chalup, Henskens and Paul, 2015).

An agent that applies Q-learning needs a number of learning episodes to find an optimal solution. An episode is a learning period that starts from a selected state and ends when a goal

state is reached. During an episode, the agent chooses an action a from the set of actions A of its current state s based on its selection policy. The learner then perceives the new state of the environment s', and receives a reward $R(s,a)$ based on the previously implemented action. The agent then updates its Q-table based on the Q-function:

$$Q(s,a) \longleftarrow (1-\alpha)\, Q(s,a) + \alpha\, [R(s,a) + \gamma\, max_{a' \in A} Q(s',a')], \text{where } s \in S, a \in A,$$
$$\alpha \in [0,1] \text{ is the learning rate, and } \gamma \in [0,1] \text{ is the discount factor.} \quad (2.1)$$

This procedure repeats until the agent reaches the goal state, which marks the end of the episode. The main output of the Q-learning algorithm is a policy $\pi : S \rightarrow A$ which maximises the sum of its rewards $R = r_0 + \gamma r_1 + \ldots + \gamma^n r_n$ for an MDP that has a terminal state s_t, or a termination condition. The fact that Q-learning does not require a model of the environment is one of its strengths (Strehl, Li, Wiewiora, Langford and Littman, 2006).

2.2 Second Stage: Sharing of Q-values

This section discusses the second learning stage (learning by interaction) of the following cooperative Q-learning algorithms: BEST-Q, AVE-Q, PSO-Q, WSS, and an algorithm that aggregates their results.

BEST-Q, AVE-Q and PSO-Q evaluate their Q-values using a method to approximate the rewards (Iima and Kuroe, 2008; Iima et al., 2010; Di Mario et al., 2013). In this method, the Q-values that have been updated during one episode are evaluated by adding the rewards after they are discounted. This estimation can be calculated as follows:

$$E = \sum_{K=1}^{L} \gamma^{L-K} r_K \quad (2.2)$$

where L is the number of actions performed during the episode, r_K is the reward for the k-th action, and γ is the discount factor. γ is used to balance between the rewards obtained in the beginning with those obtained in the end.

2.2.1 BEST-Q

In the second learning stage of BEST-Q (Iima and Kuroe, 2007; Abed-Alguni, 2014; Abed-alguni et al., 2015), the best Q-value for each state-action pair is selected from the Q-tables of all of the learners. Then, each learner updates its Q-table by replacing each of its Q-values with the corresponding best Q-value:

$$Q_i(s,a) \longleftarrow Q^{best}(s,a)\ (\forall i,s,a), \text{ where } i \text{ is the agent identification number.} \quad (2.3)$$

Figure 1 shows the BEST-Q algorithm. A disadvantage of this approach is that the optimal Q-values may not be found because the Q-values of all of the agents become the same after each update. BEST-Q, however, can obtain a good policy in long time simulations (Iima and Kuroe, 2006).

$Q_{best}(s,a)$: best state-action value for action a in state s.
E: evaluated value for Q-values updated.
E^{best}: evaluated value for the best Q-values.
Step 1 Set the initial values of $Q_i(s,a)(\forall i,s,a)$, and set $t \leftarrow 0$.
Step 2 Perform the following procedures.
Step 2-1 Set $i^* \leftarrow 1$ and $E^{best} \leftarrow -\infty$.
Step 2-2 Set $y \leftarrow 1$.
Step 2-3 Update $Q_{i^*}(s,a)$ by performing Q-learning for one episode for agent i^*.
Step 2-4 Calculate the evaluated value E for $Q_{i^*}(s,a)$ using Eq. 2.2.
Step 2-5 If $E > E^{best}$, set $Q^{best}(s,a) \leftarrow Q_{i^*}(s,a)(\forall s,a)$ and $E^{best} \leftarrow E$.
Step 2-6 If $y < Y$, set $y \leftarrow y+1$ and return to **Step 2-3**.
Step 2-7 If $i^* = I$, go to **Step 3**. Otherwise, set $i^* \leftarrow i^*+1$ and return to **Step 2-2**.
Step 3 Set $Q_i(s,a) \leftarrow Q^{best}(s,a)(\forall i,s,a)$.
Step 4 Set $t \leftarrow t+IY$. If $t \geq T$, terminate this algorithm. Otherwise, return to **Step 2**.

Figure 1: The BEST-Q Algorithm (Iima and Kuroe, 2006).

2.2.2 AVE-Q

AVE-Q (Iima and Kuroe, 2008; Abed-Alguni, 2014) is similar to BEST-Q except that each learner updates its Q-values by averaging its current Q-value and the best Q-value for each state-action pair from the Q-tables of all of the learners:

$$Q_i(s,a) \longleftarrow \frac{Q^{best}(s,a)+Q_i(s,a)}{2}\ (\forall i,s,a),\ \text{where } i \text{ is the agent's identification number.} \tag{2.4}$$

The AVE-Q algorithm has a similar structure to the BEST-Q algorithm in Figure 1 except that the equation in **Step 3** should be replaced with Equation 2.4. A problem with AVE-Q is that it does not eliminate of the bad Q-values at the interaction stage. Instead, it moves into the middle of its current value and the best value found so far which may lead to an incorrect policy (Abed-Alguni, 2014).

2.2.3 PSO-Q

The PSO-Q algorithm (Iima and Kuroe, 2006; Abed-Alguni, 2014) uses Particle Swarm Optimisation (PSO) to find a global optimal solution. PSO is an optimisation method that repetitively improves a candidate solution according to a qualitative measure (Kennedy and Eberhart, 1995). PSO solves decision problems that have multiple decision variables. In PSO, a swarm is a collection of particles (candidate solutions) (Iima and Kuroe, 2007). Let D be a decision problem of n decision variables $y = \{y_1, y_2, ..., y_n\}$ that minimise an objective function $f(y)$, and the size of the swarm for D be composed of m particles $\{p^1, p^2, ..., p^m\}$. The particle p^i of the swarm at iteration t is $y^i(t) = (y_1^i(t), y_2^i(t), ..., y_n^i(t))$. The following two functions

determine the candidate solution of p^i at the next iteration $t+1$:

$$v_j^i(t+1) \leftarrow wv_j^i(t) + c_1 r_1[p_j^i(t) - y_j^i(t)] + c_2 r_2[g_j(t) - y_j^i(t)]. \tag{2.5}$$

$$y_j^i(t+1) \leftarrow y_j^i(t) + v_j^i(t+1). \tag{2.6}$$

Where $v^i(t) = \{v_1^i(t), v_2^i(t), ..., v_n^i(t)\}$ is the velocity vector of p^i at iteration t, w, c_1, and c_2 are weight parameters, r_1, and r_2 are random numbers between 0 and 1, p^i is the personal best of particle i, j is the subscript for the n decision variables, $g_i(t)$ is the best solution found by particle i at iteration t and $g(t) = \{g_1(t), g_2(t), ..., g_n(t)\}$ is the best solution found by all particles since the beginning to iteration t.
In PSO-Q, the RL problem is modelled as an optimisation problem in which the solution candidates are Q-values, and the qualitative measure is the Q-function. In PSO-Q, the best Q-values of each learner and the best global Q-values of all learners are used by each learner to update its Q-table. Based on equations 2.5 and 2.6, the update equations for the RL problem can be modelled as (Iima and Kuroe, 2007):

$$V_i(s,a) \leftarrow WV_i(s,a) + C_1 R_1[P_i(s,a) - Q_i(s,a)] + C_2 R_2[G(s,a) - Q_i(s,a)]. \tag{2.7}$$

$$Q_i(s,a) \leftarrow Q_i(s,a) + V_i(s,a). \tag{2.8}$$

Where V_i is the velocity of agent i for state-action pair (s,a), W, C_1 and C_2 are weight parameters and R_1 and R_2 are random numbers between 0 to 1. Figure 2 shows the PSO-Q algorithm.

Two issues may degrade the performance of PSO-Q (Abed-Alguni, 2014). First, the success of this method depends on the initial values of its weight parameters (W, C_1, C_2) which may be different for every tested problem. This may require multiple simulations of a specific problem to determine suitable values for the weight parameters. Second, PSO-Q may not search outside the neighbourhood of the best Q-value for each state-action pair for all agents ($G(s,a)$ in Figure 2).

2.2.4 WSS

In WSS (Ahmadabadi and Asadpour, 2002; Ahmadabadi et al., 2006), each learner assigns a weight to the Q-tables of each other learner based on each learner's relative expertness. Then, each learner uses the weighted average of all of the Q-table values to update its own Q-table:

$$Q_i(s,a) \longleftarrow \sum_{j=1}^{n} (W_{ij} Q_j(s,a)) \tag{2.9}$$

Where W_{ij} is the weight learner i assigns to learner j's expertness. There are several expertness measures that have been shown to have similar outcomes when used with WSS

V_i(s, a): difference of Q_i(s, a) before and after its update.
P_i(s, a): best state-action value for action a in state s found by agent i so far.
G(s, a): best state-action value for action a in state s found by all agents so far.
E_i evaluated value for P_i(s, a).
E^G evaluated value for G(s, a).
W, C_1, C_2: weight parameters.
R_1, R_2 uniform random numbers in the range from 0 to 1.
Step 1 Set the initial values of Q_i(s, a) and V_i(s, a) ($\forall i$, s, a), and set t $\leftarrow$ 0, $E_i \leftarrow -\infty$ ($\forall i$) and $E^G \leftarrow -\infty$.
Step 2 Perform the following procedures.
Step 2-1 Set i* $\leftarrow$ 1.
Step 2-2 Set y $\leftarrow$ 1.
Step 2-3 Update Q_{i*}(s, a) by performing Q-learning for one episode for agent i*.
Step 2-4 Calculate the evaluated value E for $Q_{i*}(s, a)$ using Eq. 2.2.
Step 2-5 If E $>$ E_{i*}, set P_{i*}(s, a)$\leftarrow$ Q_{i*} (s, a) ($\forall s$,a) and $E_{i*} \leftarrow$ E.
Step 2-6 If E $>$ E^G, set G(s, a) $\leftarrow$ Q_{i*}(s, a)($\forall s$,a) and $E^G \leftarrow$ E.
Step 2-7 If y $<$ Y, set y $\leftarrow$ y $+$ 1 and return to **Step 2-3**.
Step 2-8 If i*= I, go to Step 3. Otherwise, set i* $\leftarrow$ i* $+$ 1 and return to **Step 2-2**.
Step 3 Update V_i(s, a) and Q_i(s, a) ($\forall i$, s, a) by V_i(s, a) $\leftarrow$ WV_i(s, a) $+C_1R_1$(P_i(s, a) -Q_i(s, a)) $+C_2R_2$(G(s, a) -Q_i(s, a)), and Q_i(s, a) $\leftarrow$ Q_i(s, a) + V_i(s, a).
Step 4 Set t $\leftarrow$ t + IY. If t $\geq$ T, terminate this algorithm. Otherwise, return to **Step 2**.

Figure 2: The PSO-Q Algorithm (Iima and Kuroe, 2006).

(Ahmadabadi and Asadpour, 2002). One of these measures is the Normal measure (Nrm) which is defined as the sum of the rewards that a learner receives during the individual learning stage (Ahmadabadi and Asadpour, 2002):

$$x_i^{Nrm} = \sum_{t=1}^{now} r_i(t) \tag{2.10}$$

Where $r_i(t)$ is the reward of agent i at instant t. The formula for assigning a weight to agent j's knowledge by learner i, when using the knowledge of all agents is:

$$W_{ij} \longleftarrow \frac{x_i}{\sum_{k=1}^{n} x_k} \tag{2.11}$$

Where n is the number of agents, and x_k is the expertness of agent k for $k = 1, ..., n$. Several weight assigning mechanisms can be found in Ahmadabadi and Asadpour (2002) and Ahmadabadi et al. (2006). A disadvantage of WSS is that the optimal Q-values may not be found when the Q-values vary to a large degree when the Q-values are shared (Abed-Alguni, 2014). This is because outlier Q-values will distort the average Q-value.

2.2.5 Aggregate Sharing Strategy

The cooperative Q-learning algorithms BEST-Q, AVE-Q, PSO-Q, and WSS exhibit different behaviour based on their sharing strategies. In addition, none of these algorithms has proven superiority over the other cooperative algorithms for all problems. These observations motivated us to combine each of the sharing strategies into a single sharing strategy in an attempt to reduce the variability in performance for different problems. In the rest of this paper, the Q-learning algorithm that uses this strategy is referred to as average aggregation Q-learning.
In average aggregation Q-learning, each agent updates its Q-values by averaging the Q-values from each of the BEST-Q, AVE-Q, PSO-Q, and WSS algorithms for each state-action:

$$Q_i(s,a) \longleftarrow \frac{Q^{BEST-Q}(s,a) + Q^{AVE-Q}(s,a) + Q^{WSS}(s,a) + Q^{PSO-Q}(s,a)}{4} \tag{2.12}$$

$(\forall i, s, a)$ where i is the agent's identification number, and 4 is the number of the sharing strategies.

3 Modern Approaches

This section discusses modern approaches to Q-learning that speed up the learning process. The cooperative Q-learning approaches studied in this paper use a simple selection policy during individual Q-learning. The approaches described in this section enhance Q-learning by modifying the selection policy to improve performance. Thus, future research may be able to combine these enhancements with cooperative Q-learning to achieve even greater improvements.
A significant problem in RL is the trade-off between *exploitation* of actions that have been found to be highly rewarded and *exploration* of actions that have not yet been tested (Chen, Lin, Tan and Shi, 2009; Louie, 2013). Bayesian Q-learning (BQL) is a variation of the Q-learning algorithm that suggests a solution to balance between exploration and exploitation (Vlassis, Ghavamzadeh, Mannor and Poupart, 2012). BQL uses Bayes' rule to represent the uncertainty about Q-values. This is accomplished by maintaining a probability distribution over the Q-values and choosing the actions that maximise the expected long-term reward based on this distribution.
For example (Guez, Silver and Dayan, 2012), let $\mathcal{T} : S \times A \times S \to \mathbb{R}$ (where S is the set of states and A is the set of actions) be an MDP. The dynamics of $\mathcal{T}$ are typically unknown. Therefore, $\mathcal{T}$ is normally assumed to be a latent variable distributed according to the probability distribution $P(\mathcal{T})$. At any instant t, the history of actions and states ($h_t = s_1 a_1 s_2 a_2 ... a_{t-1} s_t$) of the MDP is used to update the posterior belief on $\mathcal{T}$ using Bayes' rule $P(\mathcal{T}|h_t) \propto P(h_t|\mathcal{T})P(\mathcal{T})$. The uncertainty about the transitions of the environment can be changed using Bayes' rule into uncertainty about the current state in an augmented state space ($S^+ = S \times \mathcal{H}$, where $\mathcal{H}$ is the set of possible histories). The transitions related to S^+ can be defined as:

$$\mathcal{T}^+(< s, h >, a, < s', h' >) = \mathbf{1}[h' = has'] \int_{\mathcal{T}} \mathcal{T}(s,a,s') P(\mathcal{T}|h)\, d\mathcal{T}, \;\; \mathcal{R}^+(< s, h >, a) = R(s,a),$$

where R is the reward function and $\mathcal{R}^+$ is the reward function of S^+. (3.1)

The tuple $< \mathcal{S}^+, A, \mathcal{T}^+, \mathcal{R}^+ >$ forms what is known as the Bayes-Adaptive MDP (BAMDP) of the original MDP model. The dynamics of BAMDP are known, meaning it can be solved to obtain the value function of each action by finding the maximum expected discounted reward over policy π ($\max_{\pi} \mathbb{E}_{\pi}$):

$$Q^*(< s_t, h_t >, a) = \max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t'=t}^{\infty} \gamma^{t-t'} \, r_{t'} \, | a_t = a \right] \tag{3.2}$$

After deriving the optimal action for each state, the greedy selection policy can be used in the original MDP to select the actions. These actions form the best actions for the Bayesian agent with respect to its prior belief over T.

A major problem of BQL is that it requires much more computation to select actions and update Q-values than the classical Q-learning algorithm because the search space of BQL is enormous (Dearden, Friedman and Russell, 1998). Guez et al. (2012) addressed this problem by proposing a tractable method called Bayes-adaptive Monte-Carlo Planning (BAMCP). BAMCP is a method that utilises Monte Carlo tree search (MCTS) in which, at each iteration, a single MDP that simulates an episode is sampled from the agent's beliefs (States, Actions, Rewards, Q-values). The outcome of the simulation process of an episode is used to update the value of each node of the search tree traversed during the episode. The optimal values of each future sequence of nodes is obtained after the simulation of many sampled MDPs. In BAMCP, three sample methods can be used to get a computationally tractable algorithm: root sampling where beliefs are sampled at the beginning of each episode, lazy sampling where only the transition probabilities that are required to simulate an episode are generated, and a model-free RL algorithm that learns a rollout policy. Although BAMCP requires less computation time than BQL, it is still requires an enormous number of computations compared to conventional Q-learning. This is because MCTS requires a large number of computations to converge to optimality.

MAXQ is a hierarchical RL (HRL) algorithm that decomposes the value function of an MDP into an additive combination of smaller value functions of smaller MDPs that collectively form the MDP. MAXQ does not balance between exploration and exploitation of actions because it does not take into account the probabilistic prior knowledge of the agent about the problem space (Dietterich, 2000). Cao and Ray (2012) addressed this issue by proposing an approach to HRL that incorporates Bayesian priors in the MAXQ algorithm. This approach extends the MAXQ algorithm by incorporating priors on the primitive environment model and on goal pseudo-rewards. Priors are statistical information of previous policies and problem models that can help a reinforcement agent to accelerate its learning process. In multi-goal RL, priors can be extracted from models or policies of previous learned goals that should be given in advance in order to incorporate them in the learning process.

Ngo, Ngo and Wolfgang (2014) proposed a Bayesian based RL algorithm that uses hierarchical action decomposition to make the BQL algorithm computationally tractable. In this approach, the problem model is formulated as a partially observable semi MDP (POSMDP) that is partitioned into a hierarchy of POSMDP subtasks: lower-level subtasks and higher-level subtasks. Each POSMDP subtask is sampled from the prior belief of the environment, then solved using

Monte Carlo Value Iteration with Macro-Actions solver. This means that the learning process is an offline process because the prior belief of the environment should be given in advance.
Inverse RL (IRL) is the problem of finding the reward function of a MDP when the dynamics and the model of the environment and the behaviour of an expert are known in different circumstances. Ramachandran and Amir (2007) modelled the IRL problem as a Bayesian problem where the actions of the expert are used to update the prior of the reward functions. This posterior is used to determine an estimation of the reward function. A main advantage of this approach is that it neither requires the expert to exhibit an idle behaviour nor does it require a complete optimal policy as input to the IRL agent.
In cooperative multiagent systems, the main goal is to accomplish a shared goal by coordinating the actions of agents. Such a goal is normally hard to achieve because each agent has a specific view of the environment and has no control over the actions of its peers. Therefore, each agent forms a local policy based on its local information. Teacy, Chalkiadakis, Farinelli, Rogers, Jennings, McClean and Parr (2012) proposed a decentralised BQL approach for coordination and learning in distributed cooperative multiagent systems. In this approach, the environment is viewed as a Factored MDP (FMDP) which is an MDP with a state space S that can be specified as a cross-product of sets of state variables $S = S_0 \times S_1 \times ... \times S_{n-1}$. The proposed approach uses the Value of Perfect Information (VPI) to perform exploration for each sub-problem associated with a state variable. VPI is defined as the expected gain in reward received when the agent learns the true value of choosing action a in state s. The algorithm was tested using an Unmanned Aerial Vehicle problem, and the experimental results demonstrated that the proposed approach is efficient. However, the approach was evaluated in a small scale problem. The approach, therefore, has to be evaluated in a large scale distributed problem to show its efficiency.
A Partially Observable MDP (POMDP) is a type of MDP in which the agent cannot determine with full reliability its current state. Instead, it makes an observation based on the action and resulting state. Ross, Pineau, Chaib-draa and Kreitmann (2011) introduced the BQL algorithm to POMDPs. This new framework is based on a new mathematical model, called Bayes-Adaptive POMDP (BAPOMDP), that maintains a posterior over both the state and model parameters of the POMDP. In BAPOMDP, the posterior is updated online, each time the agent performs an action and gets an observation. However, the size of the state space of a BAPOMD can be intractable, therefore the researchers proposed an approximate algorithm for tracking beliefs in the BAPOMDP model. This algorithm reduces the infinite state space to a finite state space as it preserves the value function of the BAPOMDP to random precision.The approximate algorithm suffers from some limitations. First, simultaneous extraction of both transition and observation probabilities might cause a problem when the rewards are not explicitly received through the observations or even if the reward is perceived a priori. Another limitation is that the model is limited to discrete problems.
Roth and Erev (1995) and Erev and Roth (1998) studied the ability of RL to predict data from human subject experiments. The RL model of Roth-Erev can be defined as follows (Duffy, 2006). Given N pure strategies. Player i at instant t has a propensity $q_{ij}(t)$ to play the j^{th} pure strategy. In the beginning of the game, the propensities of all of the players are equal. The

linear choice rule can be used to find the probability that player i performs strategy j at instant t as follows:

$$p_{ij}(t) = \frac{q_{ij}(t)}{\sum_{j=1}^{n} q_{ij}(t)} \tag{3.3}$$

Given that player i received reward r at instant t for strategy k and that $R(r)$ is the reward function. The following rule is used by player i to update its propensity to play action j:

$$\begin{aligned} q_{ij}(t+1) &= (1-\Phi)q_{ij}(t) + E_k\big(j, R(r)\big), \\ E_k\big(j, R(r)\big) &= \begin{cases} (1-\epsilon)R(r), & \text{if } j{=}k \\ \big(\epsilon/(N-1)\big)R(r), & \text{otherwise} \end{cases} \end{aligned} \tag{3.4}$$

Where $R(r) = r - r_{min}$, r_{min} is the smallest possible reward, Φ is a forgetting parameter that controls the effect of past experience and ϵ is an experimentation parameter.
Roth and Erev (1995) tested their RL model with different sequential games: a market game, a best-shot/weakest link game and the ultimatum bargaining game. The experimental results suggest that Roth-Erev's RL model predicts behaviours better than Nash equilibrium point predictions. In Erev and Roth (1998), the simulation results of two versions of Roth-Erev's RL model, one-parameter version ($\Phi = \epsilon = 0$) and three-parameter version, indicate that the Roth-Erev's RL model predicts or tracks experimental data better than Nash equilibrium point predictions.

4 Experiments

In this section, the shortest path problem (Bagheri, Akbarzadeh-T and Saraee, 2008; Khanbary and Vidyarthi, 2008; Iima and Kuroe, 2008) and the taxi problem (Hengst, 2002; Cao and Ray, 2012) are used to compare the performance of the five cooperative Q-learning algorithms described in Section 2. The shortest path problem is the problem of finding the shortest path between a start cell and a goal cell in a two dimensional grid. The x and y coordinates of any cell in the grid are respectively the horizontal and vertical components of the grid. The goal cell is determined before learning begins, and the start cell is chosen randomly for each learning episode. The learner can move in four directions (up, down, right, left) unless there is an obstacle or a boundary blocking the learner's way. The shortest path problem is a single-task problem, where the agent only required to complete one operation.
In the taxi domain problem (Hengst, 2002), a taxi starts at a random position in a grid world of size 5×5, finds and picks up a passenger from a source location then navigates to a destination location and drops the passenger off. In this problem, there are four pick up and drop off locations. The taxi can perform six actions: move in four directions (up, down, right, left), pick up a passenger, or drop down a passenger. If the taxi moves to a barrier or a wall cell, the location of the taxi remains unchanged. The taxi problem is a multi-task problem, with the agent first having to locate and pick up the passenger, then locate the destination and drop the passenger off.

The cooperative Q-learning algorithms described in Section 2 (BEST-Q, AVE-Q, PSO-Q, and WSS) and the single agent Q-learning algorithm were applied to three instances of the shortest path problem (different grid sizes: 5×5, 10×10, and 20×20) and one instance of the taxi domain problem. Each instance involved three learners. These grid sizes are adequate to show how the time required to find a solution grows as the size of the problem increases.
There were two variations of the experiments. In the first variation, the learners had the same experience at each learning by interaction stage, which was accomplished by having each learner complete the same number of learning episodes in the first stage of cooperative Q-learning. In the second variation, the learners had different levels of experience, which was modelled by having each learner complete a different number of episodes in each first stage of cooperative Q-learning. For example, a learner that has spent 100 episodes learning has more experience than a learner that has only spent 50.
The reward that each learner received was defined as:

$$R_{Learner}(s,a) = \begin{cases} +5.0 & \text{if it reached the goal} \\ -0.01 & \text{otherwise} \end{cases}$$

The policy to select actions was the Softmax selection policy (Sutton and Barto, 1998), which combines exploration of the environment and exploitation of current knowledge. Given state s, an agent selects action a with a probability

$$p_s(a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{b=1}^{n} e^{\frac{Q(s,b)}{T}}}, \text{ where } T \text{ is the temperature.} \quad (4.1)$$

In equation 4.1, the temperature T controls the required degree of exploration. Assuming that all Q-values are different, if T is high, the agent will choose a random action, but if T is low, the agent will tend to select the action it currently believes is best.
The values of the learning parameters for the experiments are as follows:

- The learning rate $\alpha = 0.01$, and the discount factor $\gamma = 0.9$ in the Q-learning algorithm (as in Crook and Hayes (2003)). These values ensure adequate learning after each episode, with a preference for more recent knowledge.
- A learning episode ends when the learner reaches the goal cell or after 1000 moves without reaching the goal cell.
- The temperature $T = 0.4$ in the softmax selection policy (as in Ahmadabadi and Asadpour (2002)). This value ensures a reasonable exploration/exploitation ratio.
- The expertness measure of WSS is the Nrm measure (Section 2.2.4), which performs similarly to all other tested measures.
- The weights in PSO-Q: $W = 0$, $C1 = C2 = 1$. The weights: $W = 0$, $C1 = C2 = 2.2$ have been used in Iima and Kuroe (2006). These values caused poor performance of PSO-Q when it was implemented in the experiments.

4.1 Experimental Results

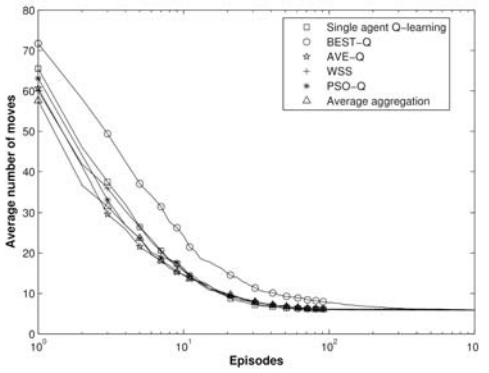
This section shows the results of the 10×10 and 20×20 grids of the shortest path problem and the 5×5 grid of the taxi problem. Results for the 5×5 grid of the shortest path problem were similar, but less interesting because of the small size of the problem.
The performance of the cooperative Q-learning algorithms and Q-learning algorithm were compared against each other following experimental models similar to those of Ahmadabadi and Asadpour (2002). In the experiments, an algorithm is said to have converged to a solution when the average number of moves to solve the problem, suggested by the learner's policy, improves by less than one over 200 consecutive episodes.

4.1.1 Equal Levels of Experience

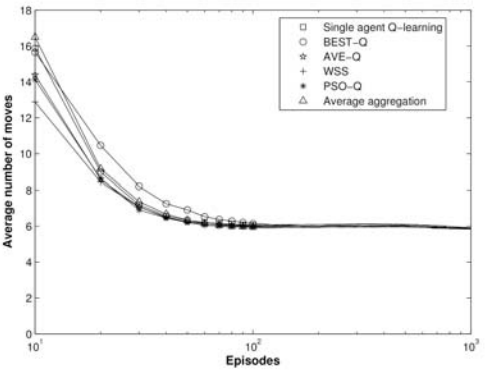
In this set of experiments, three agents learn for the same number of learning episodes before sharing their Q-values. The effects of the frequency of information sharing on the performance of cooperative learners are tested in three cases. In the first case, the agents share their Q-values after each learning episode. In the second case, the agents share their Q-values after each 10 episodes, and, in the third case, the agents share their Q-values after each 100 learning episodes.

Equal Experience for the Shortest Path Problem

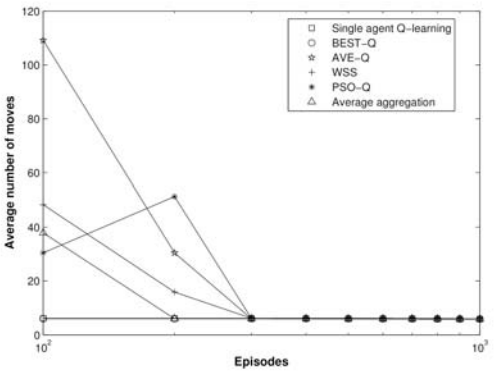
Figures 3a, 3b and 3c show the average number of steps to reach the goal cell over 1000 learning episodes in a 10×10 grid. Sharing of Q-values takes place after three periods: after each learning episode (Figure 3a), after each 10 episodes (Figure 3b), and after each 100 episodes (Figure 3c). Each of the algorithms in Figures 3a and 3b, expect BEST-Q, converges to a solution after around 180 episodes. BEST-Q requires around 420 episodes in Figure 3a and around 190 episodes in Figure 3b to converge to a solution. However, the convergence speed of BEST-Q improves with an increase in the number of the training episodes before sharing (Figure 3a: 420 episodes, Figure 3b: 190 episodes).
In Figure 3c, average aggregation, BEST-Q, and single agent Q-learning converge after around 200 episodes, while AVE-Q, PSO-Q, and WSS algorithms converge after around 340 episodes.
Figures 4a, 4b and 4c show the average number of moves to reach the goal cell in a 20×20 grid. From Figure 4a, we can see that most of the algorithms show similar performance when the learners share their Q-values after each episode. With the exception of BEST-Q, each algorithm converges to a solution after around 300 episodes. BEST-Q requires significantly more training to converge (over 1000 episodes).
The results in Figure 4b (sharing after each 10 episodes) are similar to the results in Figure 4a. In Figure 4b, each algorithm converges to a solution at around 320 episodes except BEST-Q which converges after around 420 episodes. However, BEST-Q does performs much better when sharing is after each 10 episodes than when sharing is after each episode of independent learning (less frequent sharing).
We can see from Figure 4c that each algorithm converges to a solution after around 400 episodes. AVE-Q, PSO-Q, and average aggregation initially learn slower than the other algo-



(a) Sharing Q-values after each episode.

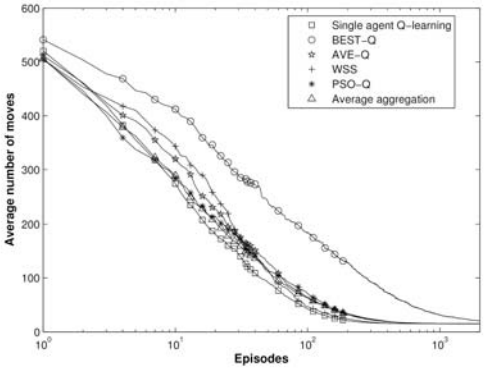


(b) Sharing Q-values after each 10 episodes.

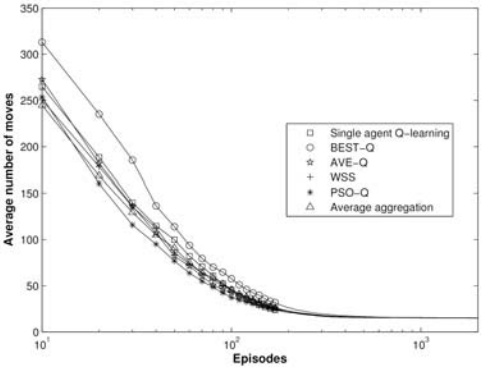


(c) Sharing Q-values after each 100 episodes.

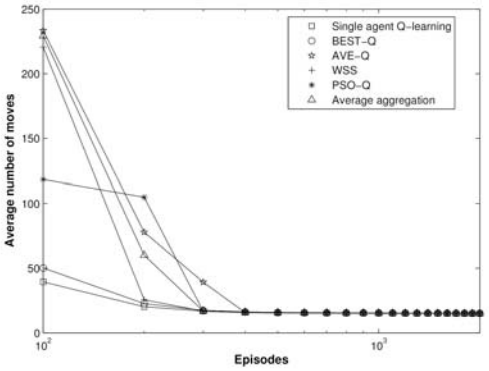
Figure 3: Average number of moves in a 10×10 grid for the shortest path problem.



(a) Sharing Q-values after each episode.

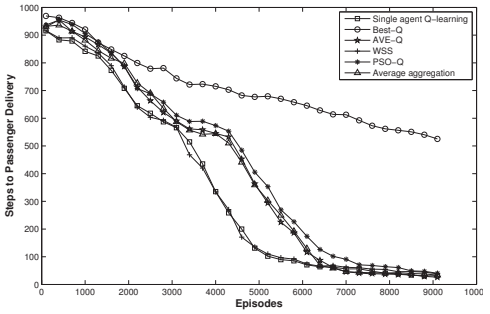


(b) Sharing Q-values after each 10 episodes.

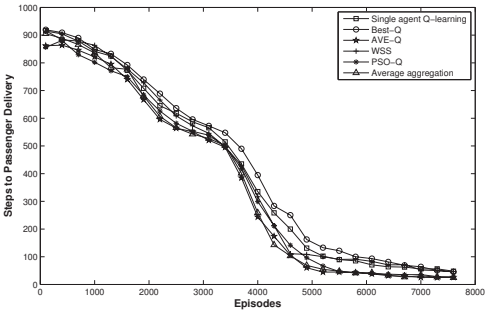


(c) Sharing Q-values after each 100 episodes.

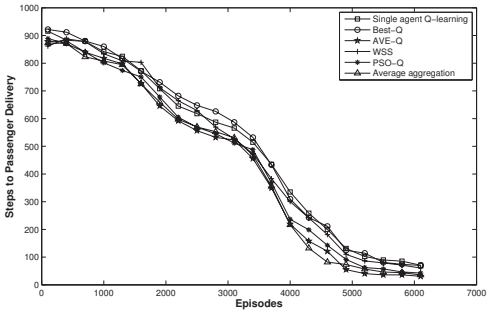
Figure 4: Average number of moves in a 20 $\times$ 20 grid for the shortest path problem.



(a) Sharing Q-values after each episode.



(b) Sharing Q-values after each 10 episodes.



(c) Sharing Q-values after each 100 episodes.

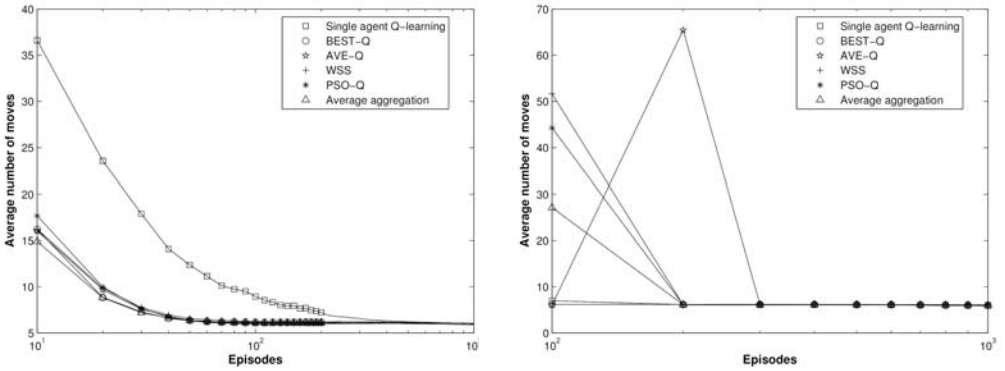
Figure 5: Average number of steps required to deliver the passenger in the taxi problem.

rithms. WSS is initially slower, but very quickly becomes comparable to BEST-Q and single agent Q-learning.

Equal Experience for the Taxi Problem

The figures in this section show the average number of steps to deliver a passenger in a 5×5 grid. Figures 5a, 5b and 5c show that most of the algorithms converge almost at the same time (after 9,200 in Figure 5a; after 7,800 in Figure 5b and after 6,200 in Figure 5c). These results indicate that the learning speed of the independent learners is only slightly affected by information sharing in the equal experience case. BEST-Q again performed poorly, but results indicate that its performance is enhanced when learners share their Q-values less frequently (did not converge after 10,000 episodes in Figure 5a; converged after 7,800 in Figure 5b and converged after 6,200 in Figure 5c).

In each of these cases, low frequency information sharing accelerates convergence compared to high frequency sharing. These results are the opposite to the experimental results of the shortest path problem in the equal experience case (Figures 3 and 4).



(a) 10, 5 and 1 episodes before sharing. (b) 100, 50 and 25 episodes before sharing.

Figure 6: Average number of moves in a 10 $\times$ 10 grid for the shortest path problem with different levels of learning.

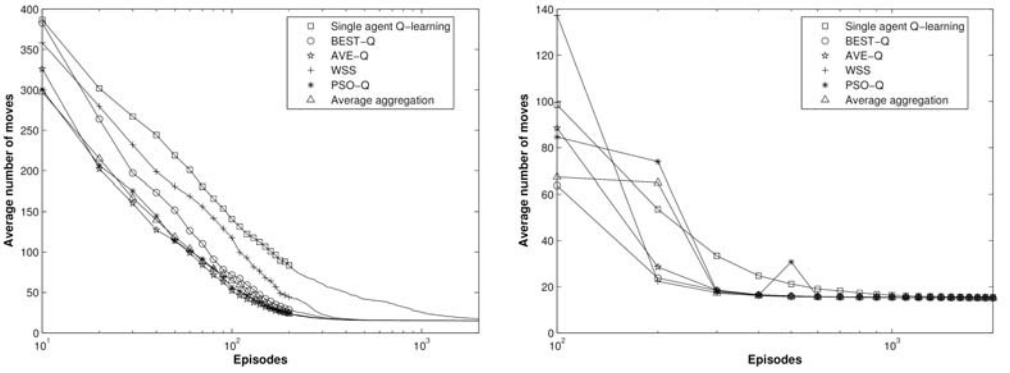
4.1.2 Different Experiences

In this set of experiments, each of the three agents learns for a different number of learning episodes in each first stage of cooperative Q-learning. The frequency of Q-value sharing is tested in two cases. In case one, the first, second, and third agents learn for 10, 5 and 1 episodes, respectively, before sharing. In case two, the first, second, and third agents learn for 100, 50 and 25 episodes, respectively. The total number of learning episodes is 1000 and the cooperation time is after the individual learning stage.

Different Experiences for the Shortest Path Problem

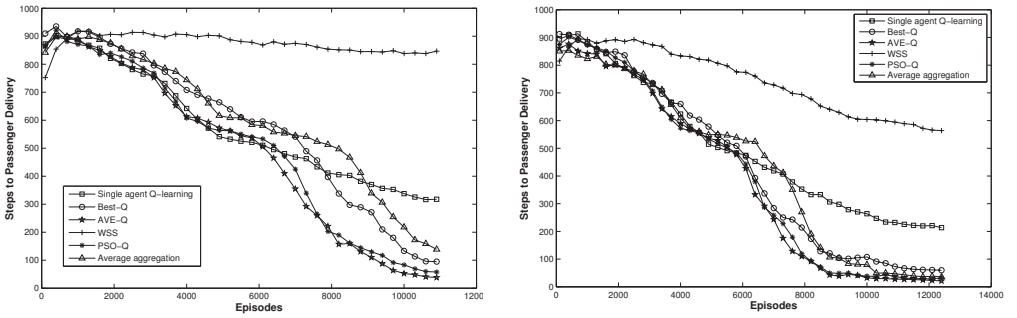
Figures 6a and 6b show the average number of steps to reach the goal cell over 1000 learning episodes in a 10×10 grid. In Figure 6a, the first, second, and third agents learn for 10, 5 and 1 episodes, respectively, before sharing. While, in Figure 6b, the agents learn for 100, 50 and 25 episodes, respectively, before sharing. Figure 6a shows that each of the cooperative Q-learning algorithms performs similarly, and better than single agent Q-learning. Figure 6b shows that single agent Q-learning and each of the cooperative Q-learning algorithms have similar performance except AVE-Q. AVE-Q learns more slowly than the other algorithms. The other average-based methods, WSS, PSO-Q, and Average aggregation, also perform worse than the other methods.

Figures 7a and 7b show the average number of steps to reach the goal cell over 2000 learning episodes in a 20×20 grid. In Figure 7a, the first, second, and third agents learn for 10, 5 and 1 episodes, respectively, before sharing. We can see from Figure 7a that each of the cooperative Q-learning algorithms has similar performance and converges to a solution faster than single agent Q-learning. In Figure 7b, the first agent learns for 100 episodes, the second learns for 50 episodes, and the third learns for 25 episodes before they share their Q-values. The figure shows that each of the cooperative Q-learning algorithms has similar performance and converges faster to a solution than single agent Q-learning. However, the performance of the



(a) 10, 5 and 1 episodes before sharing. (b) 100, 50 and 25 episodes before sharing.

Figure 7: Average number of moves in a 20 $\times$ 20 grid for the shortest path problem with different levels of learning.



(a) 10, 5 and 1 episodes before sharing. (b) 100, 50 and 25 episodes before sharing.

Figure 8: Average number of steps required to deliver the passenger in the taxi problem with different levels of learning.

PSO-Q algorithm during the first 600 learning episodes is slower than the other cooperative Q-learning algorithms. The algorithms that rely on averaging (i.e. WSS, AVE-Q, and Average aggregation) also perform worse than the other methods.

Different Experiences in The Taxi Problem

This section presents the results of the cooperative Q-learning algorithms in a 5×5 grid taxi problem. In Figure 8a, the first, second, and third agents learn for 10, 5 and 1 episodes, respectively, before sharing. In Figure 8b, the first agent learns for 100 episodes, the second learns for 50 episodes, and the third learns for 25 episodes before they share their Q-values. Figures 8a and 8b show that each of the cooperative Q-learning algorithms except WSS converges to a better solution than single agent Q-learning. WSS performs worse than single agent Q-learning. PSO-Q and AVE-Q perform the best among all the algorithms for this prob-

lem.

5 Critical Analysis

The overall results of the experiments suggest that, when all learners have an equal level of experience, sharing of Q-values has little effect on their learning speed (Section 4.1.1). In fact, in some cases, sharing Q-values actually degrades performance (Figures 3a, 3b, 4a, 4b, 5a and 5b). This is because all learners have had the same opportunities to learn, and thus are less likely to have further information than any other learner. In contrast, when learners have different levels of experience, sharing Q-values can dramatically increase learning speed, and even lead to convergence to significantly improved solutions when compared to single agent Q-learning (Most of the algorithms in Figures 7a, 7b, 8a and 8b). Inexperienced learners are able to take advantage of the increased experience of other learners to quickly progress their own learning processes.

When cooperative Q-learning is used, results also seem to improve with more frequent sharing of Q-values (e.g, Figures 8a and 8b). The most notable exception was the taxi problem when all learners had the same levels of experience (Figures 5a, 5b and 5c). This may be because the taxi problem is a multi-task problem, and optimising a single subtask may degrade the overall solution. In contrast, the shortest path problem includes a single task, and all improvements lead to a better solution overall.

Comparing individual cooperative Q-learning algorithms, BEST-Q performed worse than even single agent Q-learning in the equal experience cases (Figures 3a, 3b, 4a, 4b, 5a and 5b). A problem with BEST-Q is that the Q-values of all of the agents become the same after each sharing session which means that the optimal Q-values might not be found (Iima and Kuroe, 2006). This problem is more extreme when the learners share their Q-values so frequently that the individual learners have little chance to improve their own Q-values. However, the performance of BEST-Q can be enhanced by allowing the learners to train for a sufficient number of episodes before sharing their Q-values.

Performance of the average-based methods (AVE-Q, PSO-Q, WSS, and Average aggregation) is variable (e.g, Figures 4c and 7b vs Figures 7a and 7b), and they often converge more slowly than other algorithms when Q-values are shared less frequently (e.g, Figures 4c, 7b). This is because, at each learning by interaction stage, AVE-Q only moves halfway between its current value and the best value found so far, PSO-Q relies heavily on its initial parameters and may not search outside its global best neighbourhood, and WSS sometimes leads to wrong moves if the learner's Q-values vary widely (leading to worse performance than even single agent Q-learning for the taxi problem). The average aggregation method is guaranteed to not have the worst performance among the sharing methods, since it utilises each of them (Figures 3 to 8). However, this does require a large number of computations compared to the other algorithms.

The main contribution of this work is a comparison of existing sharing methods for Q-learning. This is the first such comparison, and demonstrates that none of the approaches perform better than all others in every situation. The average aggregation method provides some stability, guaranteed never to be the worst sharing method, though at the cost of increased computation

time. It is recommended that, for any new problem, each of the different sharing approaches should be tested to see which performs better for the given problem instance. If that is not possible, then the average aggregation method may prove beneficial.

6 Conclusion

This paper presented a comparison study of the performance of some well-known cooperative Q-learning algorithms (BEST-Q, AVE-Q, PSO-Q, and WSS) and an algorithm that aggregates their results. These algorithms were compared in two cases: when each of the learners had equal levels of expertness, and when they had different levels of expertness. The comparison study also examined the effects of the frequency of Q-value sharing on the learning speed of independent learners.

The experimental results indicate that sharing of Q-values is not beneficial when all agents have similar levels of experience, producing similar results to single agent Q-learning. However, when agents have different levels of experience, results can improve significantly. Further, more frequent sharing of Q-values can speed up these results, offering definite advantages over single agent Q-learning.

Unfortunately, some of the algorithms work very well for one problem, but quite poorly for another. Average aggregation Q-learning can be used when there is uncertainty about the benefit of using a certain sharing strategy. This method gives an overall picture of all sharing strategies and is guaranteed to not have worse performance than each of the individual methods. However, this algorithm does require a large number of calculations, since it relies on the results of the other sharing methods.

Cooperative Q-learning attempts to converge to an optimal solution by having multiple agents share their knowledge as they continue to learn. In contrast, many modern approaches to Q-learning work by optimising the selection policy of an individual learner to improve performance. Future work combining cooperative Q-learning with more modern approaches, such as BAMCP, MCTS, or BAPOMDP, may lead to even greater improvements than either individual method alone.

References

Abed-alguni, B. H., Chalup, S. K., Henskens, F. A. and Paul, D. J. (2015). A multi-agent cooperative reinforcement learning model using a hierarchy of consultants, tutors and workers, *Vietnam Journal of Computer Science* **2**(4): 213–226.

Abed-Alguni, B. H. K. (2014). *Cooperative reinforcement learning for independent learners*, PhD thesis, Faculty of Engineering and Built Environment, School of Electrical Engineering and Computer Science, The University of Newcastle, Australia.

Ahmadabadi, M., Imanipour, A., Araabi, B., Asadpour, M. and Siegwart, R. (2006). Knowledge-based extraction of area of expertise for cooperation in learning, *International Conference on Intelligent Robots and Systems, 2006 IEEE/RSJ*, pp. 3700–3705.

Ahmadabadi, M. N. and Asadpour, M. (2002). Expertness based cooperative Q-learning, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics* **32**(1): 66–76.

Bagheri, A., Akbarzadeh-T, M.-R. and Saraee, M.-H. (2008). Finding shortest path with learning algorithms, *International Journal of Artificial Intelligence* **1**(A08): 86–95.

Campbell, J. S., Givigi, S. N. and Schwartz, H. M. (2015). Multiple model q-learning for stochastic asynchronous rewards, *Journal of Intelligent & Robotic Systems* pp. 1–16.

Cao, F. and Ray, S. (2012). Bayesian hierarchical reinforcement learning, *in* F. Pereira, C. Burges, L. Bottou and K. Weinberger (eds), *Advances in Neural Information Processing Systems 25*, Curran Associates, pp. 73–81.

Chen, K., Lin, F., Tan, Q. and Shi, Z. (2009). Adaptive action selection using utility-based reinforcement learning, *in* T. Y. Lin, X. Hu, J. Xia, T.-P. Hong, Z. Shi, J. Han, S. Tsumoto and X. Shen (eds), *IEEE International Conference on Granular Computing, GRC '09.*, Nanchang, China, pp. 67–72.

Crook, P. and Hayes, G. (2003). Could active perception aid navigation of partially observable grid worlds?, *in* N. Lavra, D. Gamberger, H. Blockeel and L. Todorovski (eds), *Machine Learning: ECML 2003*, Vol. 2837 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 72–83.

Cunningham, B. and Cao, Y. (2012). Non-reciprocating sharing methods in cooperative q-learning environments, *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 02*, IEEE Computer Society, pp. 212–219.

Dearden, R., Friedman, N. and Russell, S. (1998). Bayesian q-learning, *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 761–768.

Di Mario, E., Talebpour, Z. and Martinoli, A. (2013). A comparison of pso and reinforcement learning for multi-robot obstacle avoidance, *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pp. 149–156.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition, *Journal of Artificial Intelligence Research* **13**(1): 227–303.

Doğan, B. and Ölmez, T. (2015). A novel state space representation for the solution of 2d-hp protein folding problem using reinforcement learning methods, *Applied Soft Computing* **26**: 213 – 223.

Duffy, J. (2006). Agent-Based Models and Human Subject Experiments, *in* L. Tesfatsion and K. L. Judd (eds), *Handbook of Computational Economics*, Vol. 2 of *Handbook of Computational Economics*, Elsevier, chapter 19, pp. 949–1011.

Erev, I. and Roth, A. E. (1998). Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria, *The American Economic Review* **88**(4): 848–881.

Eshgh, S. M. and Ahmadabadi, M. N. (2002). An extension of weighted strategy sharing in cooperative Q-learning for specialized agents, *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, Vol. 1, IEEE, pp. 106–110.

Galindo-Serrano, A., Giupponi, L., Blasco, P. and Dohler, M. (2010). Learning from experts in cognitive radio networks: the docitive paradigm, *2010 Proceedings of the Fifth International Conference on Cognitive Radio Oriented Wireless Networks & Communications (CROWNCOM)*, IEEE, pp. 1–6.

Gilles, E. and Peroumalnaik, M. (2008). Adapted pittsburgh classifier system: Applying reinforcement learning techniques to meteorological forecasting, *International Journal of Artificial Intelligence* **1**(A08): 96–110.

Guez, A., Silver, D. and Dayan, P. (2012). Efficient bayes-adaptive reinforcement learning using sample-based search, *in* F. Pereira, C. Burges, L. Bottou and K. Weinberger (eds), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pp. 1025–1033.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with hexq, *in* C. Sammut and A. G. Hoffmann (eds), *Machine Learning: Proceedings of the Nineteenth International Conference (ICML 2002)*, Morgan Kaufmann, pp. 243–250.

Hwang, K.-S., Jiang, W.-C. and Chen, Y.-J. (2015). Model learning and knowledge sharing for a multiagent system with dyna-q learning, *Cybernetics, IEEE Transactions on* **45**(5): 964–976.

Iima, H. and Kuroe, Y. (2006). Reinforcement learning through interaction among multiple agents, *2006 Institute of Control, Automation and Systems Engineers (ICASE), and the Society of Instrument and Control Engineers (SICE) International Joint Conference*, pp. 2457–2462.

Iima, H. and Kuroe, Y. (2007). Swarm reinforcement learning algorithms -exchange of information among multiple agents, *The Society of Instrument and Control Engineers (SICE), 2007 Annual Conference*, pp. 2779–2784.

Iima, H. and Kuroe, Y. (2008). Swarm reinforcement learning algorithms based on SARSA method, *The Society of Instrument and Control Engineers (SICE) Annual Conference 2008*, pp. 2045–2049.

Iima, H., Kuroe, Y. and Emoto, K. (2011). Swarm reinforcement learning methods for problems with continuous state-action space, *2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, pp. 2173–2180.

Iima, H., Kuroe, Y. and Matsuda, S. (2010). Swarm reinforcement learning method based on ant colony optimization, *in* O. Kaynak and G. Dimirovski (eds), *2010 IEEE International Conference on Systems Man and Cybernetics (SMC)*, IEEE, pp. 1726–1733.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks, 1995*, Vol. 4, pp. 1942–1948.

Khanbary, L. M. O. and Vidyarthi, D. P. (2008). Channel allocation in cellular network using modified genetic algorithm, *International Journal of Artificial Intelligence* **3**(A09): 126–148.

Louie, K. (2013). Exploiting exploration: Past outcomes and future actions, *Neuron* **80**(1): 6 – 9.

Ngo, V. A., Ngo, H. and Wolfgang, E. (2014). Monte carlo bayesian hierarchical reinforcement learning, *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1551–1552.

Pakizeh, E., Palhang, M. and Pedram, M. (2013). Multi-criteria expertness based cooperative q-learning, *Applied Intelligence* **39**(1): 28–40.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning, *Urbana* **51**: 61801.

Ross, S., Pineau, J., Chaib-draa, B. and Kreitmann, P. (2011). A bayesian approach for learning and planning in partially observable markov decision processes, *The Journal of Machine Learning Research* **12**: 1729–1770.

Roth, A. E. and Erev, I. (1995). Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term, *Games and Economic Behavior* **8**(1): 164–212.

Strehl, A. L., Li, L., Wiewiora, E., Langford, J. and Littman, M. L. (2006). Pac model-free reinforcement learning, *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, ACM, New York, NY, USA, pp. 881–888.

Sutton, R.-S. and Barto, A.-G. (1998). *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, USA.

Teacy, W. L., Chalkiadakis, G., Farinelli, A., Rogers, A., Jennings, N. R., McClean, S. and Parr, G. (2012). Decentralized bayesian reinforcement learning for online agent collaboration, *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 417–424.

Vlassis, N., Ghavamzadeh, M., Mannor, S. and Poupart, P. (2012). Bayesian reinforcement learning, *in* M. Wiering and M. van Otterlo (eds), *Reinforcement Learning*, Vol. 12 of *Adaptation, Learning, and Optimization*, Springer Berlin Heidelberg, pp. 359–386.

Watkins, C. (1989). *Learning from Delayed Rewards*, PhD thesis, Cambridge University, Cambridge, England.

Watkins, C. and Dayan, P. (1992). Technical note: Q-learning, *Machine Learning* **8**(3): 279–292.

Zhang, B., Mao, Z., Liu, W. and Liu, J. (2013). Geometric reinforcement learning for path planning of uavs, *Journal of Intelligent & Robotic Systems* pp. 1–19.